House Price Prediction

Machine learning is frequently used in real estate and finance to estimate house prices using linear regression. Building a predictive model to anticipate the selling price of a home based on numerous aspects or properties, such as square footage, the number of bedrooms, location, and more, is the objective of this work. Here is a succinct explanation of the procedure:

1. Data collection: Compile a dataset with past statistics about homes, such as their costs and useful characteristics. This dataset will be used as your model's training set.
2. Preprocessing the data include cleaning and preparation. This involves dealing with missing data, encoding category variables, and, if necessary, scaling numerical characteristics. Preparing the data ensures that it is in an appropriate format for the linear regression model's training.
3. Choose the elements that are most important and are most likely to have an impact on home pricing. The danger of overfitting is decreased and the model's accuracy is increased with feature selection.
4. The dataset should be split into two groups: a training set and a testing set. The testing set is used to assess the performance of the linear regression model after it has been trained on the training set.
5. Create a linear regression model, a straightforward yet powerful approach for forecasting numerical values (in this case, housing prices). The target variable (home price) and the chosen characteristics are taught to have a linear relationship via the model.
6. Model training: Using the training set of data, train the linear regression model. In order to reduce the discrepancy between the model's predictions and the training set's actual housing values, the model modifies its coefficients throughout training.
7. Model Evaluation: Use the testing set to gauge the model's effectiveness. Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE) are common assessment metrics for regression projects. These measures assess how well the model's forecasts match real home price trends.
8. Model Fine-Tuning: If the performance of the model is unsatisfactory, you may improve it by modifying its hyperparameters or by experimenting with more sophisticated regression approaches.
9. Prediction: You may use the model to make predictions on fresh, unforeseen data once it has been trained and assessed successfully. You may then make educated guesses about the costs of homes that weren't part of the training set.
10. Deployment: If the model satisfies your criteria, you may integrate it into a website, an app, or a real estate pricing system to provide consumers house price estimations depending on input attributes.

Homeowners, real estate professionals, and investors can utilise house price prediction using linear regression as a useful tool to make judgements regarding real estate transactions. Although linear regression is a fundamental technique, it is important to keep in mind that more sophisticated models, including ensemble techniques or neural networks, can be used for increased accuracy in practical situations.

```
import numpy as np
import pandas as pd
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
pd.set_option('display.max_columns',None)
```

```
train_dataset=pd.read_csv('/content/train.csv')
test_dataset=pd.read_csv('/content/test.csv')
train_dataset.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotCon
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Ins

```
print(train_dataset.shape)
print(test_dataset.shape)
```

```
(1460, 81)
(1459, 80)
```

4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	F

```
trainrow=train_dataset.shape[0]
testrow=test_dataset.shape[0]
```

```
data=pd.concat((train_dataset,test_dataset)).reset_index(drop=True)
data=data.drop('Id',1)
data.head()
```

```
<ipython-input-73-5ca61f282138>:2: FutureWarning: In a future version of pandas all arguments of DataFram
  data=data.drop('Id',1)
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig
0	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside
1	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2
2	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside
3	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corner
4	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR2

```
feature_with_na=[]
for feature in data:
    if data[feature].isnull().sum()>=1:
        feature_with_na.append(data[feature].name)
feature_with_na
```

```
['MSZoning',
 'LotFrontage',
 'Alley',
 'Utilities',
 'Exterior1st',
 'Exterior2nd',
 'MasVnrType',
 'MasVnrArea',
 'BsmtQual',
 'BsmtCond',
 'BsmtExposure',
 'BsmtFinType1',
 'BsmtFinSF1',
 'BsmtFinType2',
 'BsmtFinSF2',
 'BsmtUnfSF',
 'TotalBsmtSF',
 'Electrical',
 'BsmtFullBath',
 'BsmtHalfBath',
 'KitchenQual',
 'Functional',
 'FireplaceQu',
 'GarageType',
 'GarageYrBlt',
 'GarageFinish',
 'GarageCars',
 'GarageArea',
 'GarageQual',
 'GarageCond',
 'PoolQC',
 'Fence',
 'MiscFeature',
 'SaleType',
 'SalePrice']
```

```
for feature in feature_with_na:
    print(feature,' has ' ,np.round(data[feature].isnull().sum()/len(data[feature])*100,4),' % of missing values')
```

```
MSZoning  has  0.137  % of missing values
LotFrontage  has  16.6495  % of missing values
```

```
Alley  has  93.2169  % of missing values
Utilities  has  0.0685  % of missing values
Exterior1st  has  0.0343  % of missing values
Exterior2nd  has  0.0343  % of missing values
MasVnrType  has  0.8222  % of missing values
MasVnrArea  has  0.7879  % of missing values
BsmtQual  has  2.7749  % of missing values
BsmtCond  has  2.8092  % of missing values
BsmtExposure  has  2.8092  % of missing values
BsmtFinType1  has  2.7064  % of missing values
BsmtFinSF1  has  0.0343  % of missing values
BsmtFinType2  has  2.7407  % of missing values
BsmtFinSF2  has  0.0343  % of missing values
BsmtUnfSF  has  0.0343  % of missing values
TotalBsmtSF  has  0.0343  % of missing values
Electrical  has  0.0343  % of missing values
BsmtFullBath  has  0.0685  % of missing values
BsmtHalfBath  has  0.0685  % of missing values
KitchenQual  has  0.0343  % of missing values
Functional  has  0.0685  % of missing values
FireplaceQu  has  48.6468  % of missing values
GarageType  has  5.3786  % of missing values
GarageYrBlt  has  5.4471  % of missing values
GarageFinish  has  5.4471  % of missing values
GarageCars  has  0.0343  % of missing values
GarageArea  has  0.0343  % of missing values
GarageQual  has  5.4471  % of missing values
GarageCond  has  5.4471  % of missing values
PoolQC  has  99.6574  % of missing values
Fence  has  80.4385  % of missing values
MiscFeature  has  96.4029  % of missing values
SaleType  has  0.0343  % of missing values
SalePrice  has  49.9829  % of missing values
```

```
numerical_features=[]
for feature in data.columns:
    if data[feature].dtype!='O' and feature!= 'SalePrice':
        numerical_features.append(feature)
print('Number of Numerical Features ',len(numerical_features))

data[numerical_features].head()
```

```
Number of Numerical Features  36
```

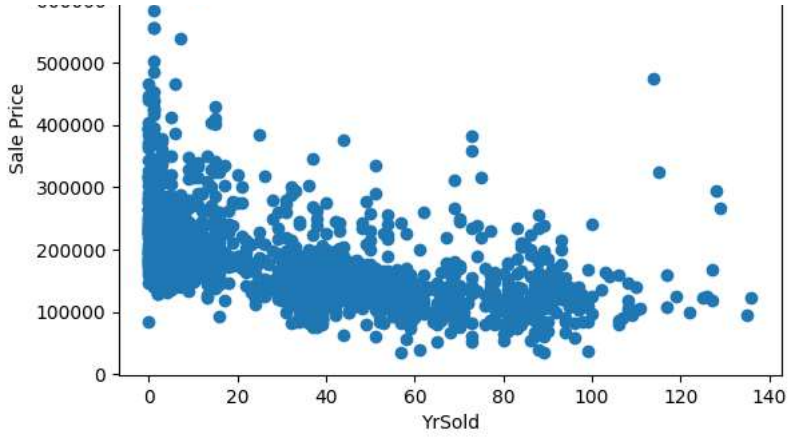
	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFi
0	60	65.0	8450	7	5	2003	2003	196.0	
1	20	80.0	9600	6	8	1976	1976	0.0	
2	60	68.0	11250	7	5	2001	2002	162.0	
3	70	60.0	9550	7	5	1915	1970	0.0	
4	60	84.0	14260	8	5	2000	2000	350.0	

```
temporal_features=[]
for feature in data.columns:
    if 'Year' in feature or 'Yr' in feature:
        temporal_features.append(feature)
data[temporal_features].head()
```

	YearBuilt	YearRemodAdd	GarageYrBlt	YrSold
0	2003	2003	2003.0	2008
1	1976	1976	1976.0	2007
2	2001	2002	2001.0	2008
3	1915	1970	1998.0	2006
4	2000	2000	2000.0	2008

```
for temp_feature in temporal_features:
    data.groupby(temp_feature)['SalePrice'].median().plot()
    plt.xlabel(temp_feature)
    plt.ylabel('Sale Price')
    plt.title(temp_feature +' vs '+' SalePrice')
    plt.show()
```

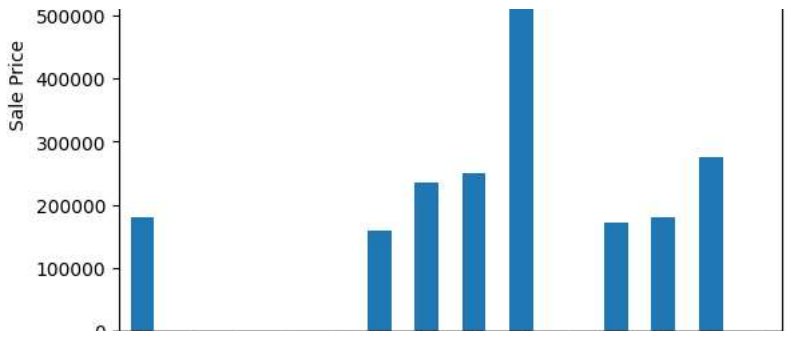
```
for i in temporal_features:
        if i!= 'YrSold':
            plt.scatter((data['YrSold']-data[i]),data['SalePrice'])
            plt.xlabel(temp_feature)
            plt.ylabel('Sale Price')
            plt.title(temp_feature +' vs '+ SalePrice')
            plt.show()
```

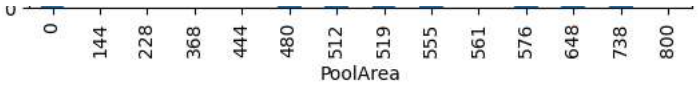


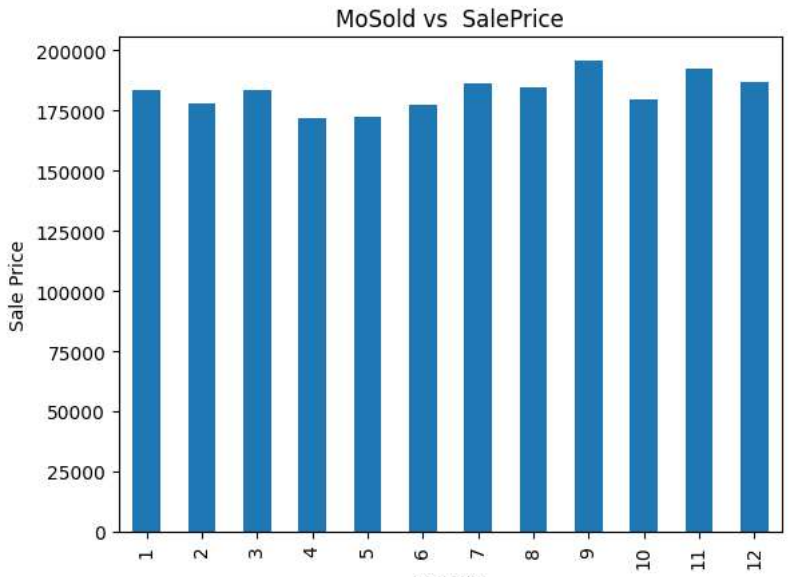

```
discrete_variables=[]
for feature in numerical_features:
    if len(data[feature].unique())<=25 and feature != 'SalePrice':
        discrete_variables.append(feature)
data[discrete_variables].head()
```

	MSSubClass	OverallQual	OverallCond	BsmtFullBath	BsmtHalfBath	FullBath	HalfBath	BedroomAbvGr	Kit
0	60	7	5	1.0	0.0	2	1	3	
1	20	6	8	0.0	1.0	2	0	3	
2	60	7	5	1.0	0.0	2	1	3	
3	70	7	5	1.0	0.0	1	0	3	
4	60	8	5	1.0	0.0	2	1	4	

```
for feature in discrete_variables:
     data.groupby(feature)['SalePrice'].mean().plot.bar()
     plt.xlabel(feature)
     plt.ylabel('Sale Price')
     plt.title(feature +' vs '+ SalePrice')
     plt.show()
```


Sale Price
500000
400000
300000
200000
100000






```
continuous_variables=[]
for feature in numerical_features:
    if feature not in discrete_variables and feature not in temporal_features:
        continuous_variables.append(feature)
data[continuous_variables].head()
```

	LotFrontage	LotArea	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFlrSF	L
0	65.0	8450	196.0	706.0	0.0	150.0	856.0	856	854	
1	80.0	9600	0.0	978.0	0.0	284.0	1262.0	1262	0	
2	68.0	11250	162.0	486.0	0.0	434.0	920.0	920	866	
3	60.0	9550	0.0	216.0	0.0	540.0	756.0	961	756	
4	84.0	14260	350.0	655.0	0.0	490.0	1145.0	1145	1053	

```
for feature in continuous_variables:
    data[feature].hist(bins=30)
    plt.title(feature)
    plt.show()
```